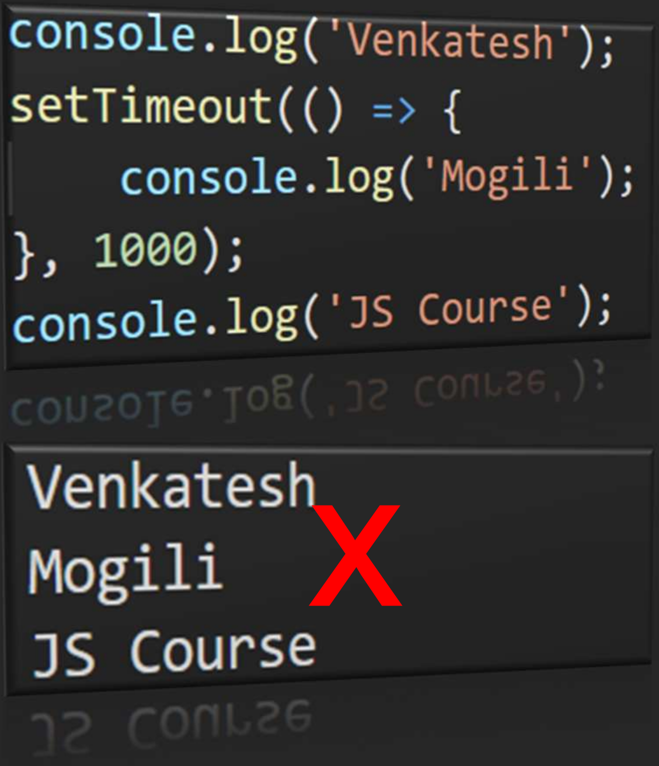
Callbacks vs Promises vs Async/Await

తెలుగు లో JavaScript

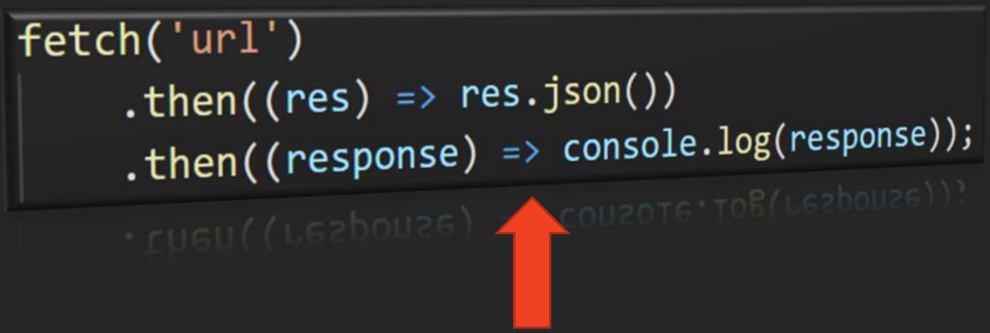
Callbacks
vs
Promises
vs
Async/Await

All are same?



```
console.log('Venkatesh');
setTimeout(() => {
    console.log('Mogili');
}, 1000);
console.log('JS Course');
```

```
Venkatesh
Mogili
JS Course
```



```
fetch('url')
    .then((res) => res.json())
    .then((response) => console.log(response));
```

Why should we use Async-Await If we already have Promises?

1

Callbacks

VENKATESH MOGILI
#Web Guru

"I will call back later!"

A callback is a function passed as an argument to another function

This technique allows a function to call another function

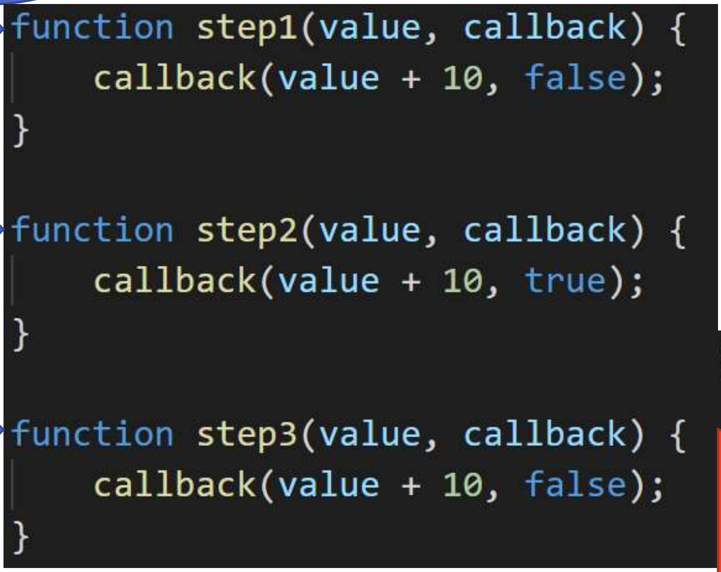
A callback function can run after another function has finished

```
setTimeout(() => {
    console.log('Hello World!');
}, 1000);
```

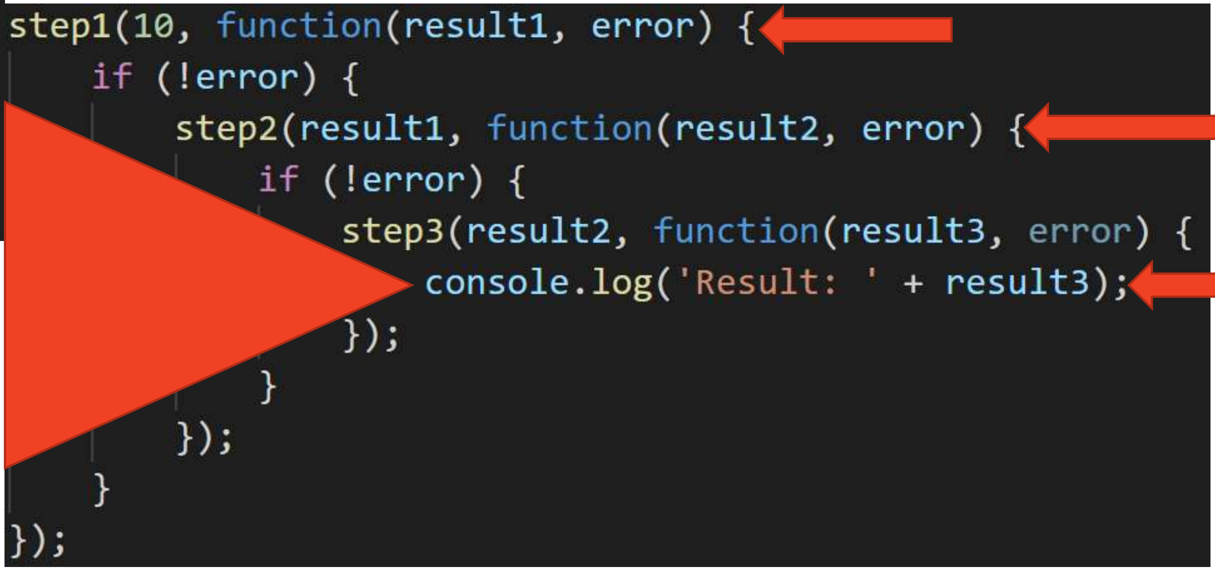
1

Callback Examples

VENKATESH MOGILI
#Web Guru



```javascript
function step1(value, callback) {
    callback(value + 10, false);
}

function step2(value, callback) {
    callback(value + 10, true);
}

function step3(value, callback) {
    callback(value + 10, false);
}
```



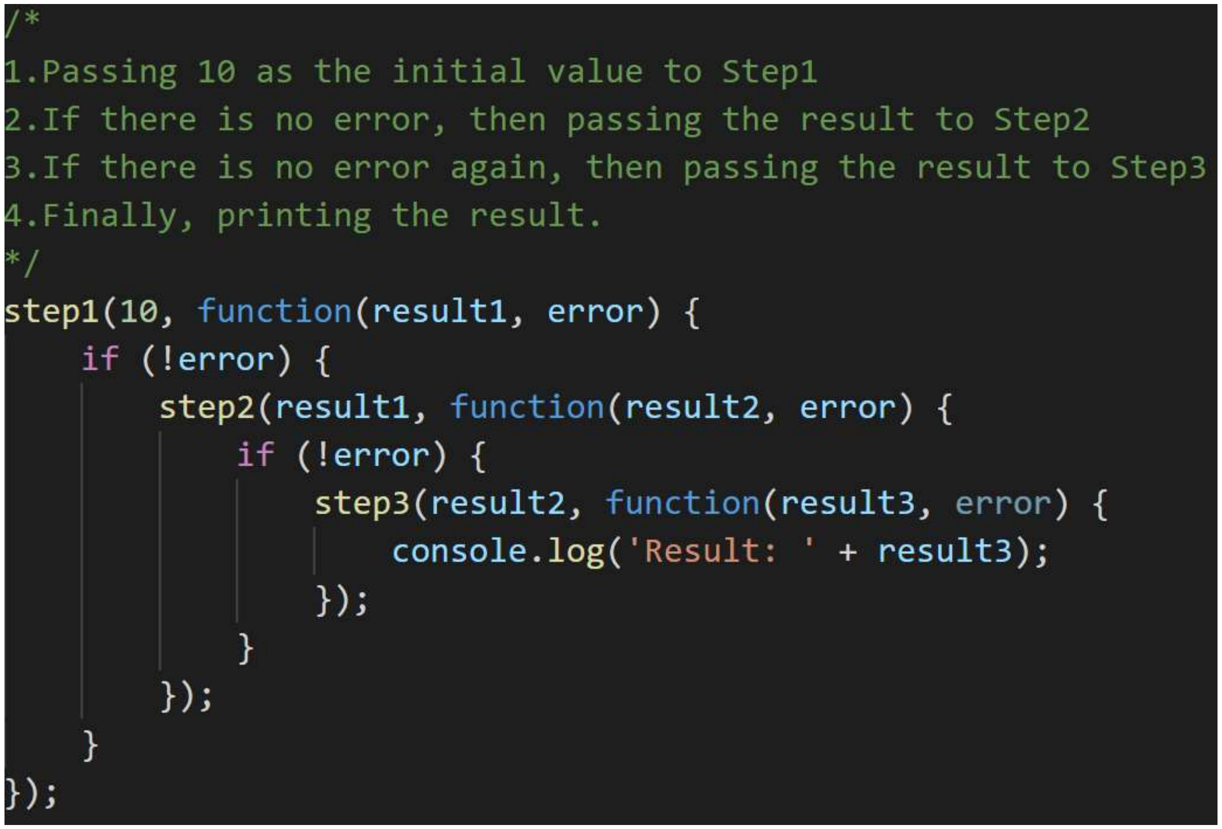
```javascript
step1(10, function(result1, error) {
    if (!error) {
        step2(result1, function(result2, error) {
            if (!error) {
                step3(result2, function(result3, error) {
                    console.log('Result: ' + result3);
                });
            }
        });
    }
});
```

1

Callback Hell Solution-1

VENKATESH MOGILI
#Web Guru

✓ Write Comments



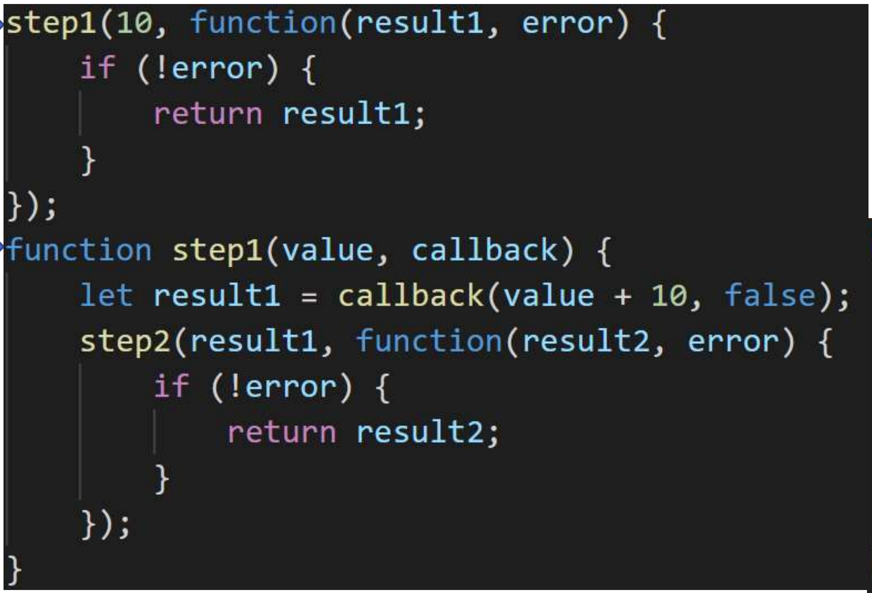
```
/*
1.Passing 10 as the initial value to Step1
2.If there is no error, then passing the result to Step2
3.If there is no error again, then passing the result to Step3
4.Finally, printing the result.
*/
step1(10, function(result1, error) {
    if (!error) {
        step2(result1, function(result2, error) {
            if (!error) {
                step3(result2, function(result3, error) {
                    console.log('Result: ' + result3);
                });
            }
        });
    }
});
```

1

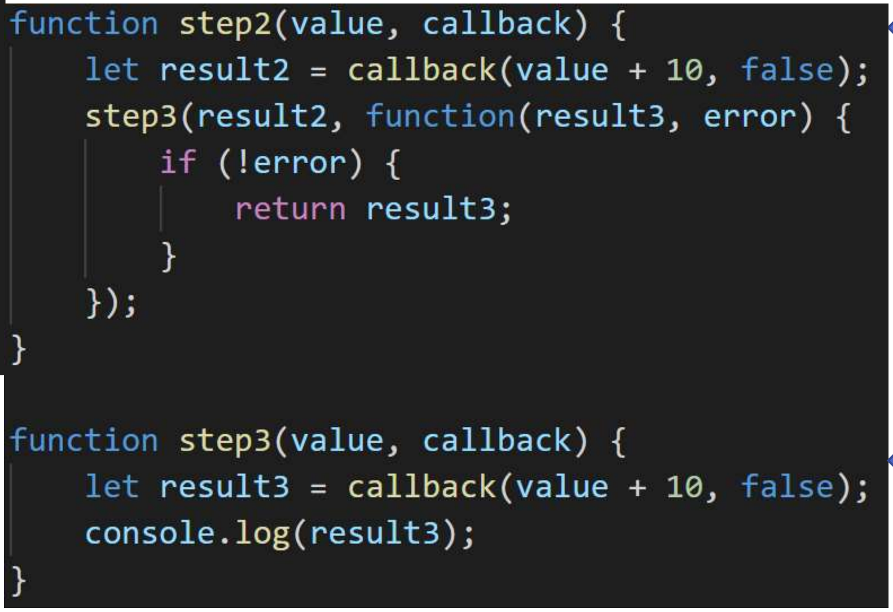
Callback Hell Solution-2

VENKATESH MOGILI
#Web Guru

✓ Split Callbacks into Smaller functions



```javascript
step1(10, function(result1, error) {
    if (!error) {
        return result1;
    }
});
function step1(value, callback) {
    let result1 = callback(value + 10, false);
    step2(result1, function(result2, error) {
        if (!error) {
            return result2;
        }
    });
}
```



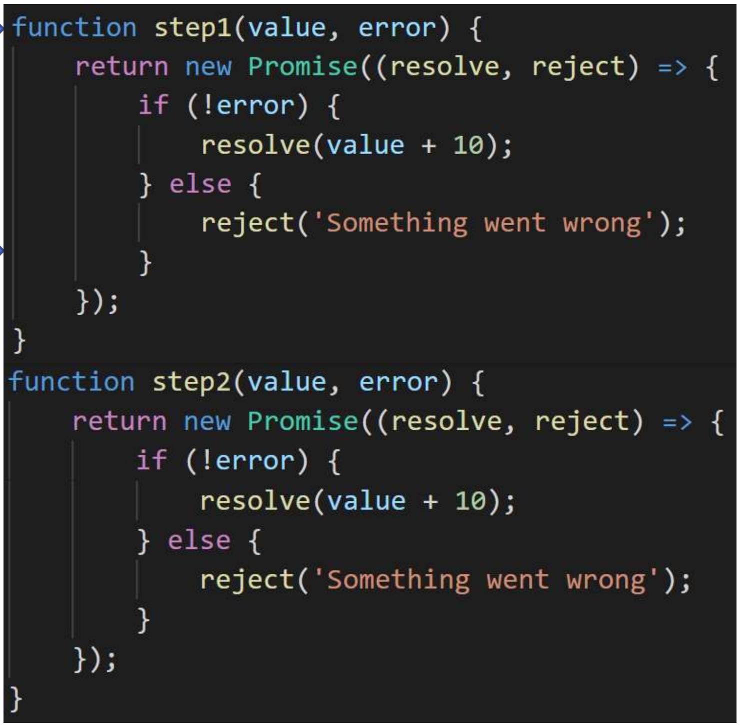
```javascript
function step2(value, callback) {
    let result2 = callback(value + 10, false);
    step3(result2, function(result3, error) {
        if (!error) {
            return result3;
        }
    });
}

function step3(value, callback) {
    let result3 = callback(value + 10, false);
    console.log(result3);
}
```

2

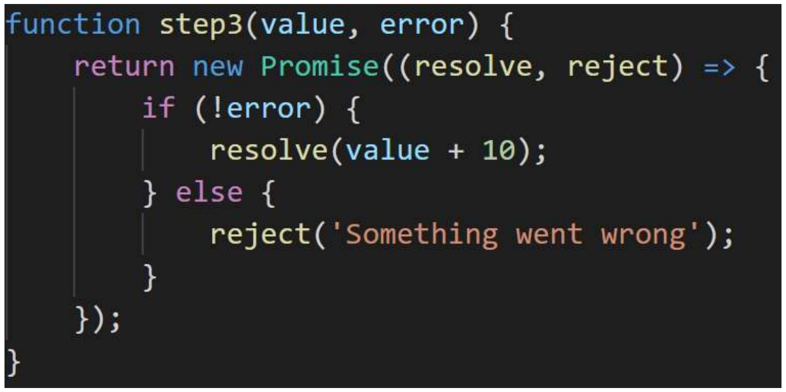
Callback Hell Solution-3

VENKATESH MOGILI
#Web Guru

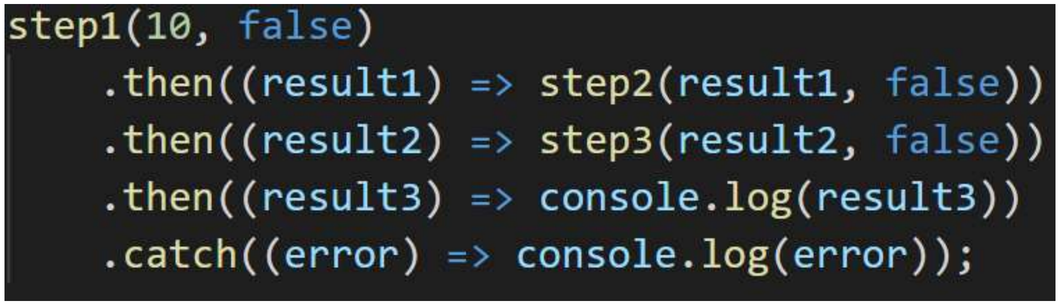
✓ Using Promises



```
function step1(value, error) {
    return new Promise((resolve, reject) => {
        if (!error) {
            resolve(value + 10);
        } else {
            reject('Something went wrong');
        }
    });
}
function step2(value, error) {
    return new Promise((resolve, reject) => {
        if (!error) {
            resolve(value + 10);
        } else {
            reject('Something went wrong');
        }
    });
}
```



```
function step3(value, error) {
    return new Promise((resolve, reject) => {
        if (!error) {
            resolve(value + 10);
        } else {
            reject('Something went wrong');
        }
    });
}
```

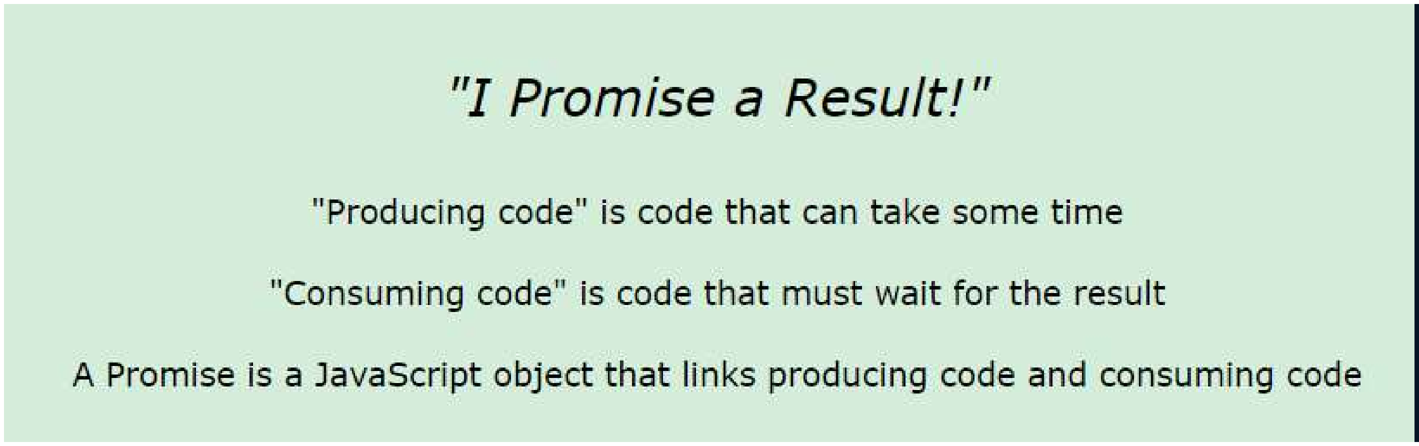


```
step1(10, false)
    .then((result1) => step2(result1, false))
    .then((result2) => step3(result2, false))
    .then((result3) => console.log(result3))
    .catch((error) => console.log(error));
```

2

Promises

VENKATESH MOGILI
#Web Guru

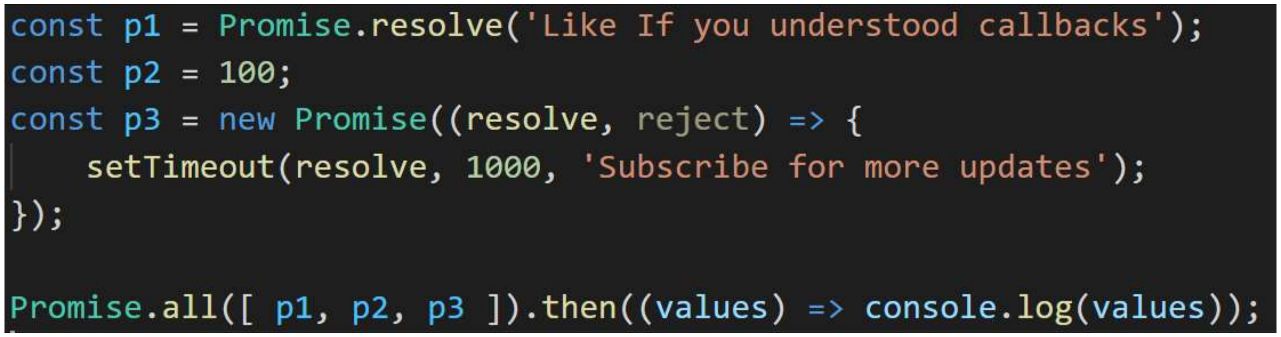


"I Promise a Result!"

"Producing code" is code that can take some time

"Consuming code" is code that must wait for the result

A Promise is a JavaScript object that links producing code and consuming code



```
const p1 = Promise.resolve('Like If you understood callbacks');
const p2 = 100;
const p3 = new Promise((resolve, reject) => {
    setTimeout(resolve, 1000, 'Subscribe for more updates');
});

Promise.all([ p1, p2, p3 ]).then((values) => console.log(values));
```

Promise.all([promises])

2

Promises

VENKATESH MOGILI
#Web Guru

Available Promise State and Values

✓ Pending	(undefined)
✓ Fullfilled	(resolved value)
✓ Rejected	(reason for rejection)

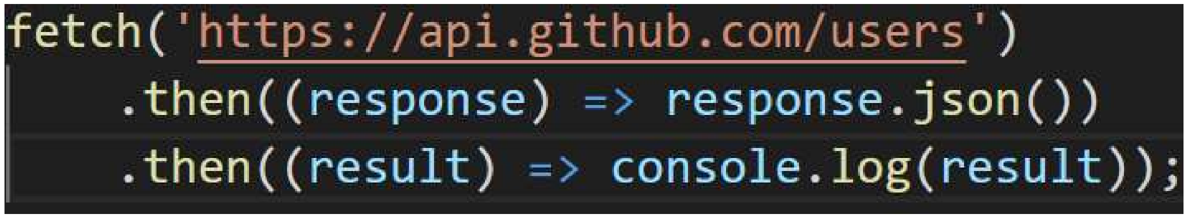
Available Promise methods

- ✓ Promise.all([promises])
- ✓ Promise.allSettled([promises])
- ✓ Promise.any([promises])
- ✓ Promise.race([promises])

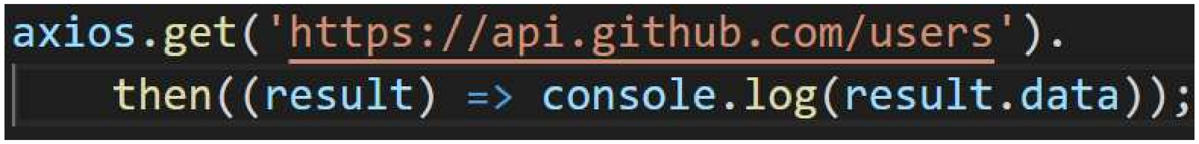
2

Promises Example

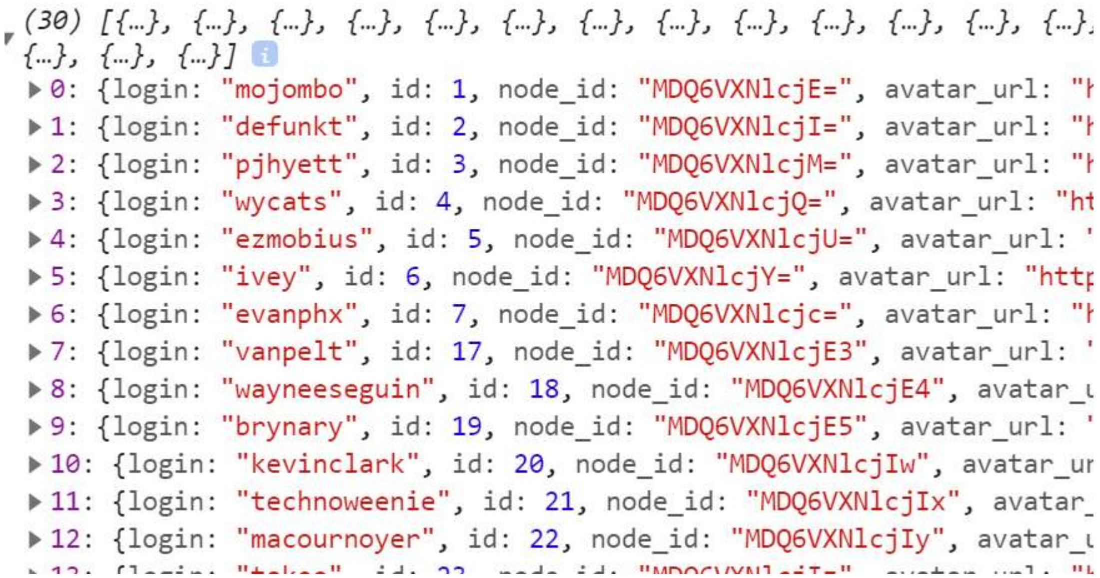
VENKATESH MOGILI
#Web Guru



```
fetch('https://api.github.com/users')
    .then((response) => response.json())
    .then((result) => console.log(result));
```



```
axios.get('https://api.github.com/users').
    then((result) => console.log(result.data));
```



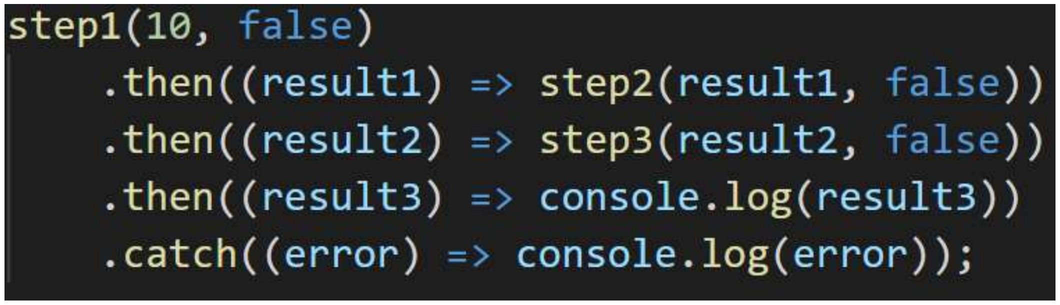
```
▾(30) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
 {…}, {…}, {…}]
 ▸0: {login: "mojombo", id: 1, node_id: "MDQ6VXNlcjE=", avatar_url: "h
 ▸1: {login: "defunkt", id: 2, node_id: "MDQ6VXNlcjI=", avatar_url: "h
 ▸2: {login: "pjhyett", id: 3, node_id: "MDQ6VXNlcjM=", avatar_url: "h
 ▸3: {login: "wycats", id: 4, node_id: "MDQ6VXNlcjQ=", avatar_url: "ht
 ▸4: {login: "ezmobius", id: 5, node_id: "MDQ6VXNlcjU=", avatar_url: '
 ▸5: {login: "ivey", id: 6, node_id: "MDQ6VXNlcjY=", avatar_url: "http
 ▸6: {login: "evanphx", id: 7, node_id: "MDQ6VXNlcjc=", avatar_url: "h
 ▸7: {login: "vanpelt", id: 17, node_id: "MDQ6VXNlcjE3", avatar_url: '
 ▸8: {login: "wayneeseguin", id: 18, node_id: "MDQ6VXNlcjE4", avatar_u
 ▸9: {login: "brynary", id: 19, node_id: "MDQ6VXNlcjE5", avatar_url: '
 ▸10: {login: "kevinclark", id: 20, node_id: "MDQ6VXNlcjIw", avatar_ur
 ▸11: {login: "technoweenie", id: 21, node_id: "MDQ6VXNlcjIx", avatar_
 ▸12: {login: "macournoyer", id: 22, node_id: "MDQ6VXNlcjIy", avatar_u
 ▸13: {login: "takeo", id: 23, node_id: "MDQ6VXNlcjIz", avatar_url: "h
```

2

What is the need of Async/Await?

VENKATESH MOGILI
#Web Guru

➢ Chaining of Promises



```
step1(10, false)
    .then((result1) => step2(result1, false))
    .then((result2) => step3(result2, false))
    .then((result3) => console.log(result3))
    .catch((error) => console.log(error));
```

3

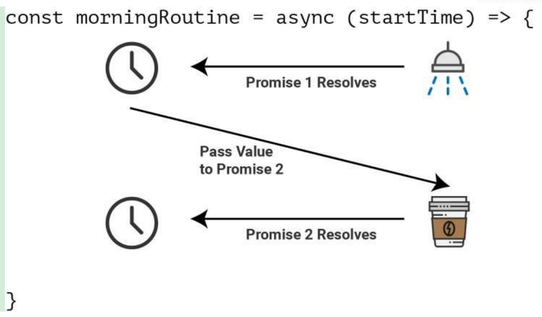
Async/Await

VENKATESH MOGILI
#Web Guru

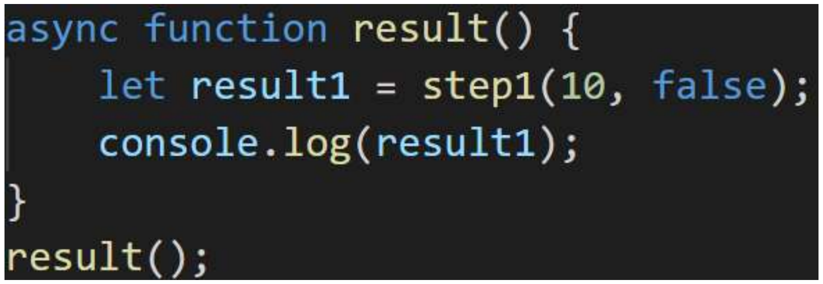
"async and await make promises easier to write"

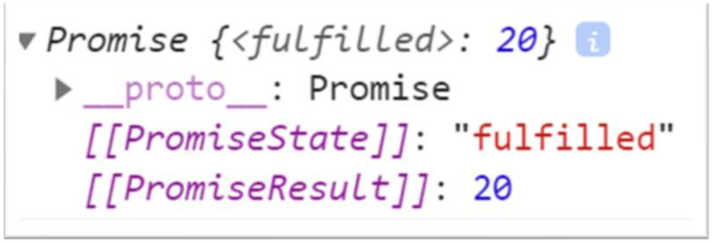
async makes a function return a Promise

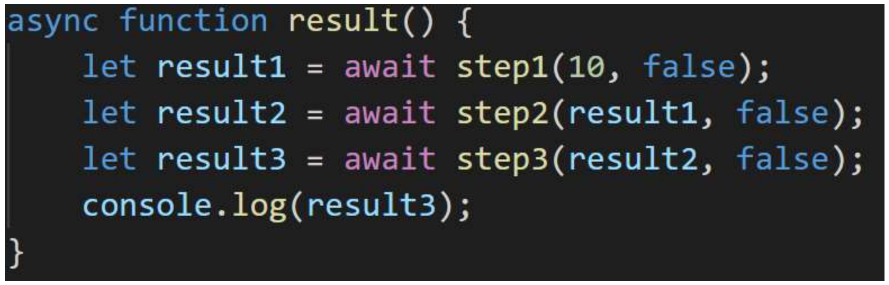
await makes a function wait for a Promise


```
const morningRoutine = async (startTime) => {
```

```
}
```


```
async function result() {
    let result1 = step1(10, false);
    console.log(result1);
}
result();
```


```
▼ Promise {<fulfilled>: 20}
  ▶ __proto__: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: 20
```


```
async function result() {
    let result1 = await step1(10, false);
    let result2 = await step2(result1, false);
    let result3 = await step3(result2, false);
    console.log(result3);
}
```

Summary

- ✓ All The 3 are same except syntax difference
- ✓ Promises have resolve , reject states
- ✓ Async function always returns a promise
- ✓ Await will take the promise and converts to actual result.

Thank you for watching

Subscribe

and

Stay Tuned

For more updates…

VENKATESH MOGILI

#Web Guru